```
// This source code is subject to the terms of the Mozilla Public License 2.0 at https://mozilla.org/M
// @@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@
// @@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@ @@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@
// @@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@    ,@@@@@@@@@@@@@@@@@@@@@@@
// @@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@      @@@@@@@@@@@@@@@@@@@
// @@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@        @@@@@@@@@@@@@@@
// @@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@         @@@@@@@@@@@@@
// @@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@           @@@@@@@@@@
// @@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@        .@@@@@@@@@@@@@@@            @@@@@@@@
// @@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@          *@@@@@@@@@@@@@@             @@@@@@@
// @@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@         @@@@@@@@@@@@@@@               @@@@@
// @@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@     @@@@@@@@@@@@@@@@                 @@@@
// @@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@                  @@@
// @@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@.                    @@
// @@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@                      @@
// @@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@.                         @
// @@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@                             @
// @@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@,                                       @
// @@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@                                                @
// @@@@@@@@@@@@@@@@@@@@@@@@@@@                                                    @
// @@@@@@@@@@@@@@@@@@@@@@@@@                                                     @@
// @@@@@@@@@@@@@@@@@@@@@@@                                                       @@
// @@@@@@@@@@@@@@@@@@@@@@                                                       @@@
// @@@@@@@@@@@@@@@@@@@@@*                @@@@@                                 @@@@
// @@@@@@@@@@@@@@@@@@@@@               @@@@@@@@@                              @@@@@
// @@@@@@@@@@@@@@@@@@@@@              @@@@@@@@@@@                           @@@@@@@
// @@@@@@@@@@@@@@@@@@@@@               @@@@@@@@%                           @@@@@@@@
// @@@@@@@@@@@@@@@@@@@@@@                                                @@@@@@@@@@
// @@@@@@@@@@@@@@@@@@@@@@@                                            @@@@@@@@@@@@@
// @@@@@@@@@@@@@@@@@@@@@@@@                                        %@@@@@@@@@@@@@@@
// @@@@@@@@@@@@@@@@@@@@@@@@@@                                   @@@@@@@@@@@@@@@@@@@
// @@@@@@@@@@@@@@@@@@@@@@@@@@@@@                           @@@@@@@@@@@@@@@@@@@@@@@@
// @@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@                @@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@
// © YinYangAlgorithms

//@version=5
indicator("Machine Learning: Optimal RSI [YinYangAlgorithms]", max_bars_back=500)
// ~~~~~~~~~~~~ INPUTS ~~~~~~~~~~~~ //
//Optimal RSI
showSignals = input.bool(true, "Show Crossing Signals", group="RSI Settings", inline="rsiToggles")
showTables = input.bool(true, "Show Tables", group="RSI Settings", inline="rsiToggles")
showNewSettings = input.bool(true, "Show New Settings", group="RSI Settings", inline="rsiToggles")
showBollingerBands = input.bool(true, "Show Bollinger Bands", group="RSI Settings", tooltip="Show RSI
optimalType = input.string("All Crossings", "Optimal RSI Type", options=["All Crossings", "Extremity C
aiAdjust = input.string("Auto", "Adjust Optimal RSI Lookback and RSI Count", options=["Auto", "Manual'
optimalLength = input.int(200, "Optimal RSI Lookback Length", maxval=500, minval=10, group="RSI Settir
rsiCount = input.int(30, "RSI Count", group="RSI Settings", maxval=50, minval=5, tooltip="How many ler
rsiMinLength = input.int(4, "RSI Minimum Length", group="RSI Settings", minval=1, tooltip="What is the
maLength = input.int(14, "RSI MA Length", group="RSI Settings", minval=1, tooltip="What length are we
backupLength = input.int(14, "Extremity Crossings RSI Backup Length", minval=1, group="RSI Settings",
//Machine Learning
useRationalQuadratics = input.bool(true, "Use Rational Quadratics", group="Machine Learning", tooltip=
```

```
onlyUseSimilarMA = input.bool(false, "Filter RSI and RSI MA", group="Machine Learning", tooltip="Shoul
useMachineLearning = input.string("Simple Average", "Machine Learning Type", options=["KNN Average", '
distanceType = input.string("Both", "KNN Distance Type", options=["Both", "Max", "Min"], group="Machir
mlLength = input.int(10, "Machine Learning Length", minval=1, group="Machine Learning", tooltip="How f
knnLength = input.int(3, "k-Nearest Neighbour (KNN) Length", minval=1, group="Machine Learning", toolt
fastLength = input.int(1, "Fast ML Data Length", minval=1, group="Machine Learning", tooltip="What is
slowLength = input.int(5, "Slow ML Data Length", minval=2, group="Machine Learning", tooltip="What is

// ~~~~~~~~~~~~ VARIABLES ~~~~~~~~~~~~ //
distanceTypeMin = distanceType == "Min" or distanceType == "Both"
distanceTypeMax = distanceType == "Max" or distanceType == "Both"
adjustRatio = aiAdjust == "Auto"

// ~~~~~~~~~~~~ FUNCTIONS ~~~~~~~~~~~~ //
//Used to refactor INPUT Settings: Optimal RSI Lookback Length and RSI Count
    //When there are bar indexs due to timeframe than having these values too high will cause the indi
autoAdjust() =>
    //Default settings (this should work on almost all Time Frames and Pairs (Premium or less Plans, n
    newLength = 90
    newCount = 20
    //total lookback can't be >= 50,000,000
        //how this is calculated is length * count * bar_index
        //IE. newLength (90) * newCount (25) * 24,000 (1 HR BTC/USDT Binance Index's available) = 54,0
        //The 50,000,000 lookback might also be based on Premium TradingView (40 second chart processi
            //need to adjust this accordingly
    //adjust based on bar_index, timeframe was an idea, but depending on what you're trading, the time
    if barstate.isrealtime
        //For realtime we no longer need to factor the Time Frame as we know how many bar_index's ther
        if bar_index <= 20000
            if bar_index <= 5000
                newLength := 200
                newCount := 30
            else if bar_index <= 10000
                newLength := 150
                newCount := 30
            else if bar_index <= 20000
                newLength := 125
                newCount := 25
            else
                newLength := 100
                newCount := 20
    else
        //For historical bar, we factor in the Time Frame as we don't know how many bar index's will b
        if bar_index <= 20000
            _timeS = timeframe.period
            _time = str.tonumber(_timeS)
            if _time >= 240 and _time < 720
                newLength := 125
                newCount := 25
            else if bar_index <= 10000 and _time >= 720 and _time < 1440
                newLength := 150
                newCount := 30
            else if bar_index <= 5000 and (str.contains(_timeS, "D") or str.contains(_timeS, "W") or s
                newLength := 200
                newCount := 30
```

```
    [newLength, newCount]

//@jdehorty Kernel Function
//used to turn a source into a rational quadratic which performs better in ML calculations
rationalQuadratic(series float _src, simple int _lookback, simple float _relativeWeight, simple int st
    float _currentWeight = 0.
    float _cumulativeWeight = 0.
    _size = array.size(array.from(_src))
    for i = 0 to _size + startAtBar
        y = _src[i]
        w = math.pow(1 + (math.pow(i, 2) / ((math.pow(_lookback, 2) * 2 * _relativeWeight))), -_relati
        _currentWeight += y*w
        _cumulativeWeight += w
    yhat = _currentWeight / _cumulativeWeight
    yhat

//Consistent and easy way to fill a table cell
f_fillCell(_table, _column, _row, _value, _color) =>
    table.cell(_table, _column, _row, _value, bgcolor = color.new(_color, 75), text_color = _color, wi

//Same as ta.sma but this way we can use a series length
pine_sma(x, series int y) =>
    sum = 0.0
    for i = 0 to y - 1
        sum := sum + x[i] / y
    sum

//Same as ta.rma but this way we can use a series length
pine_rma(src, series int length) =>
    alpha = 1/length
    sum = 0.0
    sum := na(sum[1]) ? pine_sma(src, length) : alpha * src + (1 - alpha) * nz(sum[1])

//Same as ta.rsi but this way we can use a series length
pine_rsi(x, series int y) =>
    u = math.max(x - x[1], 0) // upward ta.change
    d = math.max(x[1] - x, 0) // downward ta.change
    rs = pine_rma(u, y) / pine_rma(d, y)
    res = 100 - 100 / (1 + rs)
    res

//Calculate the optimal RSI length based on settings
getOptimalRSILength() =>
    //Storage to know the RSI and RSI MA cross percents (how much profit or loss has occured since thi
    crossPercents = array.new_float(rsiCount) //we use close not the rsi level
    //Get our source
    _src = useRationalQuadratics ? rationalQuadratic(close, 8, 8., 25) : close
    //Calculate our Optimal RSI
    for i = 0 to rsiCount - 1
        //scan all bar indexes for each RSI type to evaluate its crosses and percent increases
        len = i + rsiMinLength
        crossPercent = 0.
        crossType = 0 // 0 = none, -1 = under, 1 = over
        crossClose = 0. //close // was 0.
        crossCount = 0
```

```
            inExtremity = false
            for a = 0 to optimalLength - 1
                //get current bars RSI and MA
                currentRSI = pine_rsi(close[a], len)
                currentMA = pine_sma(currentRSI, maLength)
                //check for cross'
                crossOver = ta.crossover(currentRSI, currentMA)
                currentOver = optimalType == "All Crossings" or inExtremity ? crossOver :
                 currentRSI <= 40 and crossOver
                crossUnder = ta.crossunder(currentRSI, currentMA)
                currentUnder = optimalType == "All Crossings" or inExtremity ? crossUnder :
                 currentRSI >= 60 and crossUnder
                //calculate cross percent and update data
                if currentOver
                    if crossType != 0
                        crossPercent += crossClose / close[a]
                        crossCount += 1
                    crossClose := close[a]
                    crossType := 1
                    inExtremity := not inExtremity
                else if currentUnder
                    if crossType != 0
                        crossPercent += close[a] / crossClose
                        crossCount += 1
                    crossClose := close[a]
                    crossType := -1
                    inExtremity := not inExtremity
            //save the profit % of this RSI length
            crossPercents.set(i, crossPercent / crossCount)

        //See which RSI Length produced the highest profit
        bestPercent = -100000.
        bestIndex = 0
        for p = 0 to rsiCount - 1
            if crossPercents.get(p) > bestPercent
                bestPercent := crossPercents.get(p)
                bestIndex := p
        //get the optimal Length
        optimal = bestPercent != -100000 ? bestIndex + rsiMinLength : backupLength
        //return the best RSI Length
        [optimal, bestPercent]

    //Get the exponential average of an array, where the exponential weight is focused on the first value
    getExponentialDataAverage(_data, _length) =>
        avg = 0.
        maxLen = math.min(_data.size(), _length)
        if maxLen > 0
            for i = 0 to maxLen - 1
                curData = _data.get(i)
                tempAvg = curData
                if i > 0
                    for a = 0 to i
                        tempAvg += array.get(_data, a)
                    tempAvg := math.avg(_data.get(0), tempAvg / i)
                avg += math.avg(_data.get(0), math.avg(curData, tempAvg))
```

```
        avg / _length
    else
        avg

//Uses KNN to sort distances with our ML fast and slow data
//This is a modified version that sorts distances but rather than saving and outputting the distance a
knnAverage_fromDistance(_dataFast, _dataSlow, _minDist, _maxDist) =>
    //failsafe we need at least 1 distance
    maxDist = not _minDist ? true : _maxDist
    //Calculate the distance between slow and fast moving ML Data
    distances = array.new_float(0)
    for i = 0 to _dataSlow.size() - 1
        distance = _dataSlow.get(i) - _dataFast.get(i)
        distances.push(distance)
    //clone the array so it doesn't modify it
    clonedDistances = distances.copy()
    //slice the length from the array and calculate the max value from this slice
    splicedDistances = clonedDistances.slice(0, math.min(knnLength, clonedDistances.size()))
    maxDistanceAllowed = splicedDistances.max()
    minDistanceAllowed = splicedDistances.min()
    //scan all distances and add any that are less than max distance
    validDistances = array.new_float(0)
    for i = 0 to distances.size() - 1
        if (not maxDist or distances.get(i) <= maxDistanceAllowed) and (not _minDist or distances.get(
            distAvg = (_dataSlow.get(i) + _dataFast.get(i)) / 2
            validDistances.push(distAvg)
    // Get exponential or regular average
    if useMachineLearning == "KNN Exponential Average"
        getExponentialDataAverage(validDistances, 1)
    else
        validDistances.avg()

// ~~~~~~~~~~~~ CALCULATIONS ~~~~~~~~~~~~ //
//Adjust length and counts
if adjustRatio
    [newLength, newCount] = autoAdjust()
    optimalLength := newLength
    rsiCount := newCount
//Get our Optimal RSI Length
[rsiLength, bestPercent] = getOptimalRSILength()
//Calculate our Optimal RSI
float optimalRSI = pine_rsi(close, rsiLength)
float rsi = optimalRSI
//Calculate our temp RSI MA (this will change if our RSI does due to ML Calculations)
tempMA = ta.sma(rsi, maLength)
//Calculate if the RSI is bullish (RSI >= RSI MA) or bearish (RSI < RSI MA)
rsiBull = rsi >= tempMA

//Apply Machine Learning logic to our Optimal RSI if selected
if useMachineLearning != "None"
    if useMachineLearning == "Simple Average"
        // -- A simple machine Learning Average
            //essentially just a simple Average of Optimal RSI data (potentially filtered to account f
        rsiData = array.new_float(0)
        for i = 0 to mlLength - 1
```

```
            simpleTempMa = pine_sma(optimalRSI[i], maLength)
            simpleTempBull = optimalRSI[i] > simpleTempMa
            if not onlyUseSimilarMA or simpleTempBull == rsiBull
                rsiData.push(optimalRSI[i])
        rsi := rsiData.avg()
    else
        // -- A still simple but more complex approach to Machine Learning using KNN to sort validDist
        //Calculate our fast and slow MA's based on the current Optimal RSI
        float rsiFast = ta.sma(optimalRSI, fastLength)
        float rsiSlow = ta.sma(optimalRSI, slowLength)
        //create storage data for ML lookbacks
        rsiFastData = array.new_float(mlLength)
        rsiSlowData = array.new_float(mlLength)
        //populate our ML storage with lookbacks at our Fast and Slow Optimal RSIs
        for i = 0 to mlLength - 1
            rsiFastData.set(i, rsiFast[i])
            rsiSlowData.set(i, rsiSlow[i])
        //calculate our new Optimal RSI using KNN by using distances within KNN min/max and filtering
        rsi := knnAverage_fromDistance(rsiFastData, rsiSlowData, distanceTypeMin, distanceTypeMax)

//Calculate our RSI MA (we do this later incase the RSI changed through Machine Learning Calculations)
rsiMA = ta.sma(rsi, maLength)
//Calculate RSI and RSI MA crosses (Signals)
bullCross = ta.crossover(rsi, rsiMA)
bearCross = ta.crossunder(rsi, rsiMA)
//Bollinger Bands
upper_inner = 0.
lower_inner = 0.
upper_outer = 0.
lower_outer = 0.
if showBollingerBands
    basis = ta.sma(rsi, optimalLength)
    dev_inner = 1.6185 * ta.stdev(rsi, optimalLength)
    dev_outer = 2.0 * ta.stdev(rsi, optimalLength)
    upper_inner := basis + dev_inner
    lower_inner := basis - dev_inner
    upper_outer := basis + dev_outer
    lower_outer := basis - dev_outer

// ~~~~~~~~~~~~ PLOTS ~~~~~~~~~~~~ //
//Bollinger Bands
bb_up_outer = plot(showBollingerBands ? upper_outer : na, "Upper Band", color = color.new(color.green,
bb_up_inner = plot(showBollingerBands ? upper_inner : na, "Upper Band", color = color.new(color.green,
fill(bb_up_outer, bb_up_inner, color=color.new(color.green, 95))
bb_down_outer = plot(showBollingerBands ? lower_outer : na, "Lower Band", color = color.new(color.red,
bb_down_inner = plot(showBollingerBands ? lower_inner : na, "Lower Band", color = color.new(color.red,
fill(bb_down_outer, bb_down_inner, color=color.new(color.red, 95))
//RSI Bands
h0 = hline(70, "Upper Band", color=#787B86)
hline(50, "Middle Band", color=color.new(#787B86, 50))
h1 = hline(30, "Lower Band", color=#787B86)
fill(h0, h1, color=color.rgb(126, 87, 194, 90), title="Background")
//RSI + RSI MA and gradient fills
rsiPlot = plot(rsi, "RSI", color=#7E57C2)
plot(rsiMA, "RSI MA", color=color.yellow)
```

```
midLinePlot = plot(50, color = na, editable = false, display = display.none)
fill(rsiPlot, midLinePlot, 100, 70, top_color = color.new(color.green, 0), bottom_color = color.new(cc
fill(rsiPlot, midLinePlot, 30, 0, top_color = color.new(color.red, 100), bottom_color = color.new(colc
//Crossing Signals
plot(showSignals and bullCross ? rsi : na, color=color.green, linewidth=3, style=plot.style_circles, t
plot(showSignals and bearCross ? rsi : na, color=color.red, linewidth=3, style=plot.style_circles, tit

// ~~~~~~~~~~~~ TABLES ~~~~~~~~~~~~ //
var table perfTable = table.new(position.top_right, 4, 1, border_width = 5)
if showTables and barstate.islast
    f_fillCell(perfTable, 0, 0, "Optimal RSI Length: " + str.tostring(rsiLength), color.green)
    f_fillCell(perfTable, 1, 0, "Optimal Profit %: " + (bestPercent != -100000 ? str.tostring(math.rou
    if showNewSettings and adjustRatio
        f_fillCell(perfTable, 2, 0, "New Lookback Length: " + str.tostring(optimalLength), color.greer
        f_fillCell(perfTable, 3, 0, "New RSI Count: " + str.tostring(rsiCount), color.green)

// ~~~~~~~~~~~~ ALERTS ~~~~~~~~~~~~ //
alertcondition(bullCross, title="RSI Bull Signal", message="RSI Bull Signal")
alertcondition(bearCross, title="RSI Bear Signal", message="RSI Bear Signal")

// ~~~~~~~~~~~~ END ~~~~~~~~~~~~ //
```
